# Delimited Continuations

The Bee's Knees
Quasiconf 2012

**Andy Wingo**

## 0.1 A poll

How many of you use call/cc and continuation objects in large programs?

Do “we” really use it to implement coroutines and backtracking and threads and whatever?

Is call/cc necessary for Scheme?

## 0.2 Heresy

Those questions originally raised by racketeer Matthias Felleisen in 2000

Thesis of this presentation: call/cc bad, delimited continuations good

Felleisen has authored many papers on continuations.

Incidentally, he is also the first published author on delimited continuations.

See also http://okmij.org/ftp/continuations/against-callcc.html and http://srfi.schemers.org/srfi-18/mail-archive/msg00013.html.

## 0.3 Against call/cc (1)

Requires set! to do almost anything with multiple returns

Passing arguments to continuations: manual CPS

set! plus continuations is a common way to emulate delimited continuations

## 0.4 Against call/cc (2)

“A global goto with arguments”

Captured continuations do not compose with current continuation:

```
(call/cc (lambda (k) (k (k 1))))
```

Oleg: “Call/cc is a bad abstraction.”

In practice, if not trivial, terribly confusing.

## 0.5 Against call/cc (3)

Delimited in practice...

...but where?

Almost always too much

## 0.6 Scheme deserves better

Delimited continuations

Sitaram 1993: “Handling Control”

http://www.ccs.neu.edu/scheme/pubs/pldi93-sitaram.pdf

Felleisen 1988: “The theory and practice of first-class prompts”

http://www.cs.tufts.edu/~nr/cs257/archive/matthias-felleisen/prompts.pdf

## 0.7 Bibliography, ctd

Flatt et al 2007: "Adding Delimited and Composable Control to a Production Programming Environment."

http://www.cs.utah.edu/plt/publications/icfp07-fyff.pdf

Dybvig, Peyton-Jones, and Sabry 2007: "A monadic framework for delimited continuations"

http://www.cs.indiana.edu/~dyb/pubs/monadicDC.pdf

Academically validated!

## 0.8 Example.

```
(use-modules (ice-9 control))

(% (+ 1 (abort))   ; body
   (lambda (k) k)) ; handler
```

`%` pronounced "prompt"

What is captured:

```
(+ 1 [])
```

Wrapped in a function:

```
(lambda vals (+ 1 (apply values vals)))
```

Think tcsh for the prompt.

Guile's `abort` is Sitaram's `control`.

## 0.9 Compositional

A function, not a global goto

```
(let ((k (% (+ 1 (abort))
            (lambda (k) k))))
  (k (k 1)))
= ((lambda vals (+ 1 (apply vals vals)))
   ((lambda vals (+ 1 (apply vals vals)))
    1))
= (+ 1 (+ 1 1))
= 3
```

## 0.10 Analogy with shell

fork/exec : coredump :: `%` : `abort`

Differences

- "Cores" from delimited continuations aren't dead
- More expressive value passing
- Nestable
- The language, not the system

## 0.11 Tags

`(% tag body handler)`

```
(define-syntax-rule (let/ec k exp)
  (let ((tag (make-prompt-tag)))
    (% tag
       (let ((k (lambda args
                  (apply abort-to-prompt
                         tag
                         args))))
         exp)
       (lambda (k . vals)
         (apply values vals)))))
```

## 0.12 Optimizations

Escape-only prompts

- Handler like `(lambda (k v ...) ...)`, $k$ unreferenced
- Implementable with setjmp/longjmp, no heap allocation

## 0.13 Optimizations

Prompt elision

- `(% (make-prompt-tag) exp h)` = `exp`
- Result of inlining `(let/ec k body)`, $k$ unreferenced in body
- Provide `break`, no cost if unused

## 0.14 Optimizations

Local CPS

Fundamentally dynamic: hence “dynamic control”

## 0.15 Mental model

Aborting to escape-only prompt: longjmp

Aborting to general prompt

- Copy of stack between prompt and abort
- Copy of dynamic bindings in same

Calling delimited continuation: splat stack, augment dynamic environment

## 0.16 Other names

“Composable continuations”

“Partial continuations”

## 0.17 Other formalisms

`%` / `abort`

`%` / `control`

`call-with-prompt` / `abort-to-prompt`

`reset` / `shift`

`set` / `cupto`

All equivalent

## 0.18 Limitations

Calling a delimited continuation composes two continuations: one stays in place, the other is pushed on

No way to use copying of C stack to do this: C stack frames are not relocatable

No standard way to capture continuation without unwinding to prompt

## 0.19 But what do I do with it?

A prompt is a boundary between programs

Prompts best conceived as concurrency primitives

The REPL and your code run concurrently

## 0.20 Node with automatic CPS

Delimited continuations: the ideal building block for lightweight threads

Set file descriptors to non-blocking

If `EWOULDBLOCK`, `abort`

Scheduler installs prompt, runs processes

## 0.21 (ice-9 nio)

`nio-read`

## 0.22 (ice-9 eports)

`fdes->eport`

`file-port->eport`

`accept-eport`

`connect-eport`

`get-u8`, etc

## 0.23 (ice-9 ethreads)

`run`

`spawn`, `suspend`, `resume`, `sleep`

## 0.24 memcached-server.scm (1)

```
(define (socket-loop esocket store)
  (let loop ()
    (let ((client (accept-eport esocket)))
      (spawn (lambda ()
               (client-loop client store)))
      (loop))))
```

## 0.25 memcached-server.scm (2)

```
(define (client-loop eport store)
  (let loop ()
    (let* ((args (string-split
                  (read-line eport) #\space))
           (verb (string->symbol (car args)))
           (proc (hashq-ref *commands* verb)))
      (unless proc
        (client-error eport "Bad: ~a" verb))
      (proc eport store (cdr args)))
    (drain-output eport)
    (if (eof-object? (lookahead-u8 eport))
        (close-eport eport)
        (loop))))
```

## 0.26

## 0.27 questions?

- Guile: http://gnu.org/s/guile/
- Prompts: http://www.gnu.org/software/guile/manual/html_node/Prompts.html
- Ethreads branch: `wip-ethreads` in Guile
- Words: http://wingolog.org/
- Slides: http://wingolog.org/pub/qc-2012-delimited-continuations-slides.pdf
- Notes: http://wingolog.org/pub/qc-2012-delimited-continuations-notes.pdf